01-10-00 A

jc682 U.S. PTO
01/07/00

PATENT

Docket No. IL-10443

jc580 U.S. PTO
09/479432
01/07/00

Assistant Commissioner for Patents
Washington, DC 20231

NEW APPLICATION TRANSMITTAL

Transmitted herewith for filing is the patent application of Inventor(s):
Charles R. Musick, Terence Critchlow, Medhavan Ganesh, Thomas R. Slesak, Krzysztof Fidelis

For (title): **DATA FOUNDRY SOFTWARE**

DO#30PC-32,443,450

1. Type of Application

■ This new application is for an original patent.

□ This new application is a:
- □ Division
- □ Continuation
- □ Continuation-in-part (CIP)

2. Benefit of Prior U.S. Application(s) (35 USC 120)

□ The new application being transmitted claims the benefit of prior U.S. application(s).

3. ■ Benefit under 35 U.S.C. 119(e) of United States provisional application(s) listed below:

60/115,449	January 8, 1999
Application Serial No.	Filing Date

4. Papers enclosed which are required for filing Date Under 37 CFR 1.53(b).

16 Pages of specification
10 Pages of claims
1 Pages of abstract
10 Sheets of drawings
- ■ formal
- □ informal

5. Additional papers enclosed

□ Preliminary Amendment
□ Information Disclosure Statement
□ Form PTO-1449

6. **Declaration or oath**

- ☐ Enclosed and executed by
  - ☐ inventors
  - ☐ legal representative of inventor(s) 37 CFR 1.42 or 1.43
- ■ Not Enclosed

7. **Assignment**

- ☐ An assignment of the invention to The Regents of the University of California.
  - ☐ is attached
  - ■ will follow

8. **Certified Copy**

Certified copy(ies) of application(s)

(country)	(application no.)	(filed)
(country)	(application no.)	(filed)

from which priority is claimed

- ☐ is(are) attached.
- ☐ will follow

9. **Fee Calculation**

CLAIMS AS FILED								
Type of Claim	Number Filed	Included in Basic Fee		Number Extra		Rate		Total Fee
Total Claims	52	-20	=		x	$18	=	$ 576.00
Independent Claims	5	-3	=		x	$78	=	$ 156.00
						Multiple Claims	=	$
						Basic Filing Fee	=	$ 690.00
						Sub-Total	=	$ 1422.00
						Small Entity Filing Fee	=	**$** 711.00

10. Small Entity Statement(s)

■ Verified Statement that this is a filing by small entity under 37 CFR 1.9 and 1.27 is attached.

Filing Fee Calculation (50% of regular filing fee) $ 711.00

11. Fee Payment

□ Not Enclosed

■ Enclosed (See Account No. Below)

Total Basic Filing Fees To Be Paid **$ 711.00**

12. Method of Payment of Fees

□ Check in the Amount of $ _____

■ Charge Account No. **12-0695** in the amount of $ **711.00**

A duplicate of this transmittal is attached.

13. Instructions As To Overpayment/Underpayment

■ credit/charge
Account No. **12-0695**

□ refund

John P. Wooldridge

John P. Wooldridge
ATTORNEY
Reg. No.: 38,725

P.O. Box 808, L-703
Livermore, CA 94551
Tel. No.: (925) 424-6897

Dated: 1-7-00

DO NOT PRINT

jc682 U.S. PTO
01/07/00

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant : Charles R. Musick et al

Attorney Docket No. : IL-10443

Serial No. :

Art Unit:

Filed :

Examiner:

For : Data Foundry Software

Assistant Commissioner for Patents
Washington, D.C. 20231

09457626 - 010700

EXPRESS MAIL CERTIFICATE

"Express Mail" label number EL264587099US

Date of Deposit January 7, 2000

I hereby certify that the following *attached*

1. New Application Transmittal (in duplicate)
2. Verified Statement Claiming Small Entity Status
3. Application
   (Specification 16 pages, Claims 10 pages, Abstract 1 page)
   Ten (10) sheets of formal drawings;
4. Express Mail Certificate
5. Return postcard

is being deposited with the United States Postal Service "Express Mail Post Office to addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Assistant Commissioner for Patents, Box: Patent Application, Washington, D.C. 20231.

Nancy J. Stone
(Type or print name of person mailing paper)

(Signature of person mailing paper or fee)

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant :	Charles R. Musick et al.	Docket No. :	IL-10443
Serial No. :		Art Unit :	
Filed :		Batch No. :	
For :	Data Foundry Software	Examiner :	

09479432 - 010700

VERIFIED STATEMENT (DECLARATION) CLAIMING SMALL ENTITY STATUS [37 CFR 1.9 (f) and 1.27(d)] - NONPROFIT ORGANIZATION

I hereby declare that I am an official empowered to act on behalf of the nonprofit organization identified below:

The Regents of the University of California
1111 Franklin Street
Oakland, CA 94607-5200

TYPE OF ORGANIZATION

__X__ University or Other Institution of Higher Education

I hereby declare that the nonprofit organization identified above qualifies as a nonprofit organization as defined in 37 CFR 1.9(e) for purposes of paying reduced fees under Section 41(a) and (b) of Title 35, United States Code with regard to the invention entitled Data Foundry Software

by inventor(s) Charles R. Musick Terence Critchlow, Madhavan Ganesh, Tom R. Slezak, Krzysztof Fidelis

described in

__X__ the specification filed herewith.

___ application serial no. _____, filed _____.

___ patent no. _____, issued _____.

I hereby declare that rights under contract or law have been conveyed to and remain with the nonprofit organization with regard to the above-identified invention, except for a license to a Federal Agency pursuant to 35 USC 202(c) (4).

Each person, concern or organization to which I have assigned, granted, conveyed, or licensed or am under an obligation under contract or law to assign, grant, convey, or license any rights in the invention is listed below:

__X__ no such person, concern, or organization

I acknowledge the duty to file, in this application or patent, notification of any charge in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b))

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true: and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application, any patent issuing thereon, or any patent to which this verified statement is directed.

JANET G. TULK
Laboratory Counsel
Lawrence Livermore National Laboratory
7000 East Avenue, L-701
Livermore, CA 94550

Janet G. Tulk
JANET G. TULK

1/4/00
Date

DOCKET-UCRL-99-234

S-92,017

IL-10443

DATA FOUNDRY SOFTWARE

BY

Charles R. Musick (USA)
5345 Crown Court
Castro Valley, CA 94552

Terence Critchlow (USA)
1402 Fox Creek Court
Livermore, CA 94550

Madhavan Ganesh (India)
2459 Countrybrook
San Jose, CA 95132

Tom Slezak (USA)
3950 Pestana Way
Livermore, CA 943550

Krzysztof Fidelis (Poland)
920 Coventry Circle
Brentwood, CA 94523

DO NOT DESTROY

SYSTEM AND METHOD FOR INTEGRATING AND ACCESSING MULTIPLE DATA SOURCES WITHIN A DATA WAREHOUSE ARCHITECTURE

The United States Government has rights in this invention pursuant to Contract No. W-7405-ENG-48 between the United States Department of Energy and the University of California for the operation of Lawrence Livermore National Laboratory.

RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Application No. 60/115,449,filed January 8, 1999,entitled Datafoundry Software, which is incorporated herein by this reference.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates generally to systems and methods for data integration and management, and more particularly for integrating and accessing multiple data sources within a data warehouse architecture through such techniques as automatic generation of mediators which accept data in a specific format, perform transformations on and store the data.

Discussion of Background Art

Data warehousing is an approach for managing data from multiple sources by representing a single, consistent view it. One of the more typical data warehouse architectures, the mediated data warehouse, uses a series of data source specific wrapper and mediator layers to integrate the data into the consistent format required by the warehouse. Commercial data warehousing products have been produced by companies such as RebBrick, IBM, Brio, Andyne, Ardent, NCR, Information Advantage, Informatica, and others. Furthermore, some companies use relational databases, such as those sold by Oracle, IBM, Informix and Sybase, to develop their own in-house data warehousing solution.

These approaches are successful when applied to traditional business data because the data format used by the individual data sources tends to be rather static. Therefore, once a data source has been integrated into a data warehouse, there is relatively little work required to maintain that connection. However, that is not the case for all data sources. Some data sources, in particular within certain domains, tend to regularly change their data model, format and/or interface. This is problematic because each change requires the warehouse administrator to update the wrapper, mediator, and warehouse to properly read, interpret, and represent the new format. Because these updates can be difficult and time consuming, the regularity of data source format changes effectively limits the number of sources that can be integrated into a single data warehouse.

In order to increase the number of dynamic data sources that can be integrated into a warehouse, the cost of maintaining the warehouse must be decreased. This could be accomplished by some combination of reducing the cost to maintain the wrapper, the mediator, and the warehouse data store.

In response to the concerns discussed above, what is needed is a system and method for reducing the cost of data warehouses that integrate and provide access to multiple data sources, overcoming the problems of the prior art.

SUMMARY OF THE INVENTION

The present invention is a system and method for integrating and accessing multiple data sources within a data warehouse architecture. The system and method of the present invention are particularly advantageous over the prior art because a set of metadata is formed, providing a way to declaratively present domain specific knowledge, obtained by analyzing data sources, in a consistent and useable way. Four types of information are represented by the metadata: abstract concepts, databases descriptions, transformations and mappings.

Also, a mediator generator automatically generates data management computer code based on the metadata. The resulting code defines a translation library and a mediator class. The translation library provides a data representation for domain specific knowledge represented in a data warehouse, including "get" and "set" methods for attributes that call transformation methods and derive a value of an attribute if it is missing. The mediator class defines methods that take

"distinguished" high-level objects as input and traverse their data structures and enter information into the data warehouse.

The invention includes a method for maintaining a data warehouse, including the steps of identifying a data source of interest, updating metadata to reflect information available from the source, automatically generating a mediator based on the metadata and writing a wrapper for the source which calls the mediator. A data warehouse is defined to be any code system for integrating multiple data sources, regardless of whether the approach is based on federated database, multidatabase, or traditional warehousing technology, and independent of the computer-useable medium on which the code is stored. Metadata is defined to be equivalent to ontology. The step of updating metadata includes entering new types of information, new data formats for previously defined information, new transformations between data formats, and the schema of the source. A stand-alone mediator generation program automatically generates a fully functional mediator. An API and translation libraries are automatically defined by the mediator generation program. The wrapper makes use of the mediator. The mediator may comprise code to translate between source and target representations, possibly using externally defined methods, and load data into the warehouse. The wrapper uses the API and public data structures defined by the mediator generation program. The mediator transforms and loads data into the warehouse.

The DataFoundry metadata model includes abstractions, translations, mappings and database descriptions. The model is described by a UML

DataFoundry metadata representation, wherein the model defines the metadata used by a mediator generation program. The mediator generation program includes the steps of reading the metadata; generating translation libraries; generating an API; reading the metadata; and generating said mediator. Reading the metadata includes the steps of reading the abstraction metadata; reading the translation metadata; reading the database description metadata; and reading the mapping metadata. Translation libraries are generated by developing public and private class definitions and implementations of data structures, where the data structures comprise the abstractions and the translations.

Generating the mediator consists of creating public and private definitions and implementations of a class or classes capable of receiving data in one format, converting it to another format, and loading it into a data warehouse. Data is received by a receiving data structure defined within the translation library and is loaded into a warehouse whose schema corresponds to the database description component of the metadata. The method may be applied to a number of applications including data warehousing applications in the domain of protein sequence and structure analysis, data warehousing applications in the domain of functional genomics and proteomics, integrating a new data source into a data warehouse and updating a warehouse when a previously integrated data source is modified.

These and other aspects of the invention will be recognized by those skilled in the art upon review of the detailed description, drawings, and claims set forth below.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a dataflow diagram of a system for integrating and accessing multiple data sources within a data warehouse architecture according to the presen invention.

Figure 2 is a second dataflow diagram of the system.

Figure 3 is a dataflow diagram for metadata within the system.

Figure 4 is a relational diagram of an exemplary abstraction defined by th metadata.

Figure 5 is a relational diagram of an exemplary transformation defined by the metadata.

Figure 6 is a relational diagram of an exemplary mapping defined by the metadata.

Figure 7 is a relational diagram of an exemplary set of data structures used within the system.

Figure 8 is an Unified Modeling Language (UML) representation of the metadata schema.

Figure 9 is a relational diagram of a process used to create mediators.

Figure 10 is a dataflow diagram that gives a breakdown of the wrappers i the system.

09704040 - 022495

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Figure 1 is a dataflow diagram of a system 100 for integrating and accessing multiple data sources within a data warehouse architecture according to the present invention, and Figure 2 is a second dataflow diagram of the system. As shown by 100, a data warehouse typically has 5 layers. The data sources 102 are external to the warehouse, and contain the original data. The wrappers 104 interact with the sources to retrieve information and parse the data into an internal representation. The mediators 106 take the information, transform it into the appropriate representation and enter the data into the warehouse 108. The warehouse 108 is a large data store, often a relational database, that contains a consistent representation of a subset of data contained in the data sources 102. Users 110 interact with the data through a set of well defined interfaces. The invention is both the unique way of generating mediators 106 by using a mediator generator program 112 and our set of well-defined metadata 114.

Figure 3 is a dataflow diagram for metadata within the system. To understand the type of information required to generate a mediator 106, in Figure 3, we consider what functionality the mediator provides. The mediator takes data in a source representation 302, transforms it into a target representation 304 and then enters it into the warehouse 108. In other words, a mediator takes an abstract representation of a concrete object 306, transforms it into an alternative representation of the same concrete object 308, and maps it onto a database

description 310. Automatically generating a mediator requires representing four types of information: abstractions 902 correspond to the different representations 306 and 308; transformations 904 describe how to map between different representations 302 and 304; database descriptions 906 correspond to 310; and mappings 908 describe how the abstractions 308 relate to the database tables 310. Figure 8 presents a formalized description of the metadata in UML (Unified Modeling Language) format, a more intuitive description follows.

Figure 4 is a relational diagram of an exemplary abstraction defined by the metadata. Abstractions 902 are the core of the domain specific knowledge represented by the metadata. Conceptually, an abstraction encapsulates the different components and views of a particular domain-specific concept. Practically, an abstraction is the aggregation of all of a concept's associated attributes and representations, as presented by the participating databases. As such, the abstractions contain a superset of the information contained in any individual database.

Each abstraction inherits, directly or indirectly, from a distinguished *abstraction* class. The abstraction's attributes are optionally grouped into characteristics that combine related attributes and alternative representations of the same attribute. While this grouping has no affect on the mediator, it provides a mechanism to document the conceptual relationship between these attributes. Complex attributes can be defined in an abstraction, encouraging a natural description of the domain specific concepts. Attribute types may be primitives (i.e.

integer, string, float, etc.), structures, arrays, or pointers to an instance of another class. Each attribute has an arity associated with it, representing the number of values it can or must have.

The possible values are:

- key: the attribute is single valued, required and unique
- f_key *class*: the attribute is single valued and optional, but if it exists, its value must also occur in the *key* member of *class*
- 0: the attribute is optional and single valued. This is the default if no arity is specified.
- *num*: the attribute has exactly the number of values specified by the integer value of *num* (ex, if *num* is 1, the associated attribute is required and single valued)
- N: the attribute is optional and multi-valued
- 1_N: the attribute is multi-valued but must have at least 1 associated value

To ensure that abstractions remain a superset of the component databases, incorporating a new database requires updating them in two ways. First, any previously unknown concepts represented by the new data source must be incorporated into the class hierarchy. Second, any new representations or components of an existing abstraction must be added to its attribute list.

Database descriptions 906 are language independent definitions of the information contained within a single database. These definitions are used to identify the translations that must be performed when transferring data between a

specific data source and target. The metadata representation of a database closely mirrors the physical layout of a relational database. There are two advantages to using this independent representation of the data. First, the database attributes have the same functional expressibility as the abstraction attributes described above. As a result, they are able to represent non-relational data sources, including object-oriented databases and flat files; a crucial capability when dealing with a heterogeneous environment. Second, the ability to comment the database descriptions improves warehouse maintainability by reducing the potential for future confusion. Class comments may be used to clarify the interactions with other classes, define or refine the concept associated with a table, etc.. These comments are complimented by attribute comments that, while infrequently used for abstraction attributes, provide additional metadata about the attribute's purpose and representation.

Figure 7 is a relational diagram of an exemplary database description defined by the metadata. As databases are integrated into the warehouse, their descriptions must be entered into the metadata, as must the descriptions for all target (i.e. warehouse) databases. Furthermore, as their schemata change the database descriptions and mappings contained within the metadata must be updated to reflect these changes. These modifications are made by the database administrator (DBA). Because of the similarity between the metadata and relational formats, it is possible to automatically generate most of the database description directly from the metadata associated with most commercial DBMSs; obviously the

DBA must still explicitly enter any comments they wish to provide. However, because most flat file databases do not maintain any metadata, their metadata description must be manually defined.

Figure 5 is a relational diagram of an exemplary transformation defined by the metadata. Transformations 904 describe which attributes contain the same data, but in different formats, and identify the methods that can be used to translate between them. It is important to note that a sequence of method invocations may be required to obtain the desired representation. The metadata does not define these methods explicitly, instead it records their names, locations, and the participating attributes. These methods are restricted to operating only on class member variables and, as such, do not require any parameters. To provide the maximum flexibility, two other extensions may be associated with an abstraction, and thus shared with all its instances: class methods and class data. Class methods are called by other methods that require the ability to convert between representations, but do not require an instance of this class. Class data is useful for providing information such as a translation table that does not vary between instances of the class.

There are two benefits to identifying transformations in the metadata. First, and most obvious, it provides the final piece of knowledge required to generate the mediators. However, a subtler benefit is the combination of the transformation methods into a single library (8-5). By explicitly identifying these methods, and defining them in a single location, code re-use is encouraged and maintenance costs reduced.

Figure 6 is a relational diagram of an exemplary mapping defined by the metadata. Mappings 908 identify the correspondence between database descriptions and abstractions at both the class and attribute levels. In particular, several source classes are mapped onto a single target class to completely define an instance of the target class. When the source database is a data source 102, its classes comprise the possible source classes and the abstraction classes are the possible targets. The reverse mapping is used for the warehouse 108. Because abstractions are an aggregation of the individual databases, there is always a direct mapping between database and abstraction attributes. Due to representational differences, however, an abstraction may be split across several database classes and a single database class may be related to several abstractions. Ambiguity about which attributes should participate in the join may arise if there are multiple foreign key references in a single table. This ambiguity is resolved by explicitly identifying the join conditions in the mapping definition. Natural joins are used to combine tables when the abstraction attribute is required, otherwise outer-joins are performed.

Figure 8 defines the DataFoundry metadata model using the Unified Modeling Language (UML) . The schema of the four main components of the metadata (abstractions 902, mappings 908, database description 906, and transformation descriptions 904) are described in full detail in this UML diagram, as anyone familiar with the art will recognize. This diagram formalizes much of the previous discussion of Figures 3-7 by explicitly defining the attributes of each

DO NOT FILE - 25464.0

component of the metadata, as well as and the relationships between components and their attributes.

Figure 9 is a relational diagram of a process used to create mediators. Once the metadata 114 has been defined, the mediator generator 112 is used to generate the C++ classes and methods that comprise the mediator class 106. The mediator generator produces both a translation library 910 and a mediator class 106. The translation library represents the classes and methods associated with the abstractions 902, while the mediator classes are responsible for entering the data into the warehouse. The API available to the wrapper is a combination of the mediator class 106 and translation library 910 APIs. The process of obtaining these components from the metadata is relatively straightforward, and is therefore only discussed briefly.

Figure 10 is a dataflow diagram for accessing data from a library using the system. The translation library encapsulates the class definitions and methods associated with the domain-specific abstractions. This information is created based on the information contained in the abstraction 902 and transformation 904 metadata. The mediator generator reads this metadata into an internal representation, then outputs a distinguished *abstraction* class, and one class for each metadata abstraction. The inheritance hierarchy for the output classes is the same as the abstraction hierarchy, except that the base classes inherit from *abstraction*. *Abstraction* provides all classes with a minimal amount of functionality, including

access to both the source and target databases. The appropriate user defined extensions, as defined by the transformation metadata, are included in the class API. Class data and class methods are defined as static, class variables and methods. Transformation methods are declared as private methods – but the actual implementation is left up to the user.

The data members associated with a class correspond to the abstraction attributes; static data members are used to represent the class-data extensions. Primitive attributes types are replaced by specialized types that keep track of whether or not they have been defined. For example, attributes declared to be of type integer are recast as type mg_integer, which is a structure containing an integer value, and a boolean value *assigned*. Multi-valued abstraction attributes are represented as structures that have an additional data member, *next_ptr*, which is used to create a linked-list. Classes are also defined for complex data types, which are named based on the corresponding attribute name. For each attribute, the mediator generator defines two data access methods: one to read it (*get*), the other to write it (*put*). The get method calls appropriate translation methods in the translation library 910 to derive the value of the attribute if it is not currently available. Infinitely recursive calls are prevented by keeping track of the call stack, and not calling a method that depends on a value you are already trying to derive. Put methods set the value of attribute to be the input parameter. For multi-valued attributes, the new value is placed into the linked list of values. Because of their

DO-2000-2646-4650

complexity, the mediator generator will not produce code that invokes any of the class methods.

In addition to forming the internal representation of the mediator, the translation library, as shown in Figure 10, defines the internal data source representation 1002 used by the wrapper 104. This representation is used by the parser 1004 to represent the data from a source 102 before passing it on to the mediator 106.

Mediator class generation is only slightly more difficult than generating the translation library. For each target database schema 108, a mediator class is generated to perform the data transformations and enter the data into the warehouse. Different classes are used because the mappings vary depending on the warehouse schema, and using a pure data-driven approach to dynamically identify the appropriate transformations would be too slow. For each top-level abstraction, the generator creates a single mediator method, within the mediator class, to transfer the data contained in the abstraction instance to the warehouse. This method calls several private methods to recursive through all of the object's complex attributes and to find all possible mappings. For each method, the combination of available attributes is compared against the mapping metadata to determine if any mappings are satisfied. If a mapping becomes satisfied, code is created to enter data from the abstraction representation into the warehouse. This may require iterating over multiple values if the attributes are not single-valued. If the most recently added attribute contains attributes that reference other classes, code to continue the

recursion is generated, with each of these attributes becoming the most recently added in turn. Again, this may require the code to iterate over instance values if the attribute is multi-valued.

As databases evolve and additional data sources are integrated, new database descriptions and mappings are defined by the DBA. These may, in turn, require adding new abstractions, extending the attribute set associated with an existing abstraction, and defining new translation methods. Incorporating a new data source requires the DBA to describe it, map the source attributes to corresponding abstraction attributes, ensure that all applicable transformation methods are defined, and create the wrapper. The mediator generator creates the new mediator class, and extends the API as needed. Once a database has been integrated, adapting to schema changes often requires only modifying the wrapper to read the new format.

While the present invention has been described with reference to a preferred embodiment, those skilled in the art will recognize that various modifications may be made. Variations upon and modifications to the preferred embodiment are provided by the present invention, which is limited only by the following claims.

We claim:

1. A method for maintaining a data warehouse, comprising:

identifying a data source of interest;

updating a metadata to reflect information available from said source;

automatically generating a mediator based on said metadata; and

writing a wrapper for said source which calls said mediator.

2. The method of claim 1, wherein the step of updating a metadata comprises entering new types of information, new data formats for previously defined information, new transformations between data formats, and the schema of said source.

3. The method of claim 1, wherein said mediator is fully functional and is automatically generated by a stand-alone mediator generation program.

4. The method of claim 3, wherein said mediator generation program automatically defines an API and translation libraries

5. The method of claim 4, wherein said mediator comprises code to translate between source and target representations, possibly using externally defined methods, and load data into said warehouse.

6. The method of claim 1, wherein said wrapper makes use of said mediator.

7. The method of claim 3, wherein said mediator generation program defines a public data representation, wherein said wrapper uses said public data representation.

8. The method of claim 3, wherein said wrapper uses said mediator to load data into said warehouse.

9. A DataFoundry metadata model comprising abstractions, translations, mappings and database descriptions.

10. The model of claim 9, comprising a UML DataFoundry metadata representation.

11. The model of claim 9, wherein said model defines metadata used by a mediator generation program, wherein said mediator generation program generates a mediator.

12. The method of claim 11, wherein said mediator generation program comprises:

DOCKET NO. 950-045

reading said metadata;

generating translation libraries;

generating an API; and

generating said mediator.

13. The method of claim 12, wherein the step of reading said metadata comprises reading the abstraction metadata; reading the translation metadata; reading the database description metadata; and reading the mapping metadata.

14. The method of claim 12, wherein the step of generating translation libraries comprises developing public and private class definitions and implementations of data structures.

15. The method of claim 14, wherein said data structures comprise said abstractions and said translations.

16. The method of claim 12, wherein generating the mediator consists of creating public and private definitions and implementations of a class or classes capable of receiving data in one format, converting it to another format, and loading it into a data warehouse.

17. The method of claim 16, wherein said data is received by a receiving data structure defined within said translation library and said data is loaded into a warehouse whose schema corresponds to the database description component of the metadata.

18. The method of claim 1, wherein said method is applied to data warehousing applications in the domain of protein sequence and structure analysis.

19. The method of claim 1, wherein said method is applied to data warehousing applications in the domain of functional genomics and proteomics.

20. The method of claim 1, wherein said method is used for integrating a new data source into a data warehouse.

21. The method of claim 1, wherein said method is used for updating a warehouse when a previously integrated data source is modified.

22. The model of claim 9, as defined by the UML DataFoundry representation.

23. The method of claim 14, wherein said data structures correspond to said abstractions and said translations.

24. A computer-useable medium embodying computer program code for maintaining a data warehouse by executing the steps of:

identifying a data source of interest;

updating a metadata to reflect information available from said source;

automatically generating a mediator based on said metadata; and

writing a wrapper for said source which calls said mediator.

25. The computer-useable medium of claim 24, wherein the step of updating a metadata comprises entering new types of information, new data formats for previously defined information, new transformations between data formats, and the schema of said source.

26. The computer-useable medium of claim 24, wherein said mediator is fully functional and is automatically generated by a stand-alone mediator generation program.

27. The computer-useable medium of claim 24, wherein said mediator generation program automatically defines an API and translation libraries

09476950 - 040700

28. The computer-useable medium of claim 27, wherein said mediator comprises code to translate between source and target representations, possibly using externally defined methods, and load data into said warehouse.

29. The computer-useable medium of claim 24, wherein said wrapper makes use of said mediator.

30. The computer-useable medium of claim 26, wherein said mediator generation program defines a public data representation, wherein said wrapper uses said public data representation.

31. The computer-useable medium of claim 26, wherein said wrapper uses said mediator to load data into said warehouse.

32. A DataFoundry metadata model comprising abstractions, translations, mappings and database descriptions.

33. The model of claim 32, comprising a UML DataFoundry metadata representation.

DOCKET: 25945-0060

34. The model of claim 32, wherein said model defines metadata used by a mediator generation program, wherein said mediator generation program generates a mediator.

35. The model of claim 34, wherein said mediator generation program comprises a computer-useable medium embodying computer program code for maintaining a data warehouse by executing the steps of:

reading said metadata;

generating translation libraries;

generating an API; and

generating said mediator.

36. The model of claim 35, wherein the step of reading said metadata comprises reading the abstraction metadata; reading the translation metadata; reading the database description metadata; and reading the mapping metadata.

37. The model of claim 35, wherein the step of generating translation libraries comprises developing public and private class definitions and implementations of data structures.

38. The model of claim 37, wherein said data structures comprise said abstractions and said translations.

38. The model of claim 35, wherein generating the mediator consists of creating public and private definitions and implementations of a class or classes capable of receiving data in one format, converting it to another format, and loading it into a data warehouse.

39. The model of claim 38, wherein said data is received by a receiving data structure defined within said translation library and said data is loaded into a warehouse whose schema corresponds to the database description component of the metadata.

40. The model of claim 24, wherein said method is applied to data warehousing applications in the domain of protein sequence and structure analysis.

41. The model of claim 24, wherein said method is applied to data warehousing applications in the domain of functional genomics and proteomics.

42. The model of claim 24, wherein said method is used for integrating a new data source into a data warehouse.

43. The model of claim 24, wherein said method is used for updating a warehouse when a previously integrated data source is modified.

09450942 - 040700

44. The model of claim 32, as defined by the UML DataFoundry representation.

45. The model of claim 37, wherein said data structures correspond to said abstractions and said translations.

46. An apparatus for maintaining a data warehouse, comprising:

means for identifying a data source of interest;

means for updating a metadata to reflect information available from said source;

means for automatically generating a mediator based on said metadata; and

means for writing a wrapper for said source which calls said mediator.

47. The method of claim 1, wherein said method is applied to data warehousing applications in the domain of astrophysics and climate modeling.

48. The method of claim 1, wherein said method is applied to data warehousing applications in the domain of medical image processing and analysis.

09474952 - 070700

49. The method of claim 1, wherein said method is applied to data warehousing applications in the domain of tracking consumer and customer preferences.

50. The method of claim 1, wherein said method is applied to data warehousing applications in the domain of satellite and terrestial communication systems analysis.

51. The method of claim 1, wherein said method is used for integrating a new data source into a data warehouse.

52. The method of claim 1, wherein said method is used for updating a warehouse when a previously integrated data source is modified.

09645206 - 062702

SYSTEM AND METHOD FOR INTEGRATING AND ACCESSING MULTIPLE DATA SOURCES WITHIN A DATA WAREHOUSE ARCHITECTURE

A system and method is disclosed for integrating and accessing multiple data sources within a data warehouse architecture. The metadata formed by the present method provide a way to declaratively present domain specific knowledge, obtained by analyzing data sources, in a consistent and useable way. Four types of information are represented by the metadata: abstract concepts, databases, transformations and mappings. A mediator generator automatically generates data management computer code based on the metadata. The resulting code defines a translation library and a mediator class. The translation library provides a data representation for domain specific knowledge represented in a data warehouse, including "get" and "set" methods for attributes that call transformation methods and derive a value of an attribute if it is missing. The mediator class defines methods that take "distinguished" high-level objects as input and traverse their data structures and enter information into the data warehouse.

Meta-Data
114
Mediator Generator
112
100
API
Wrapper
Wrapper
PDB
Swiss Prot
102
Mediator
106
Data store
108
110
104

FIG. 1

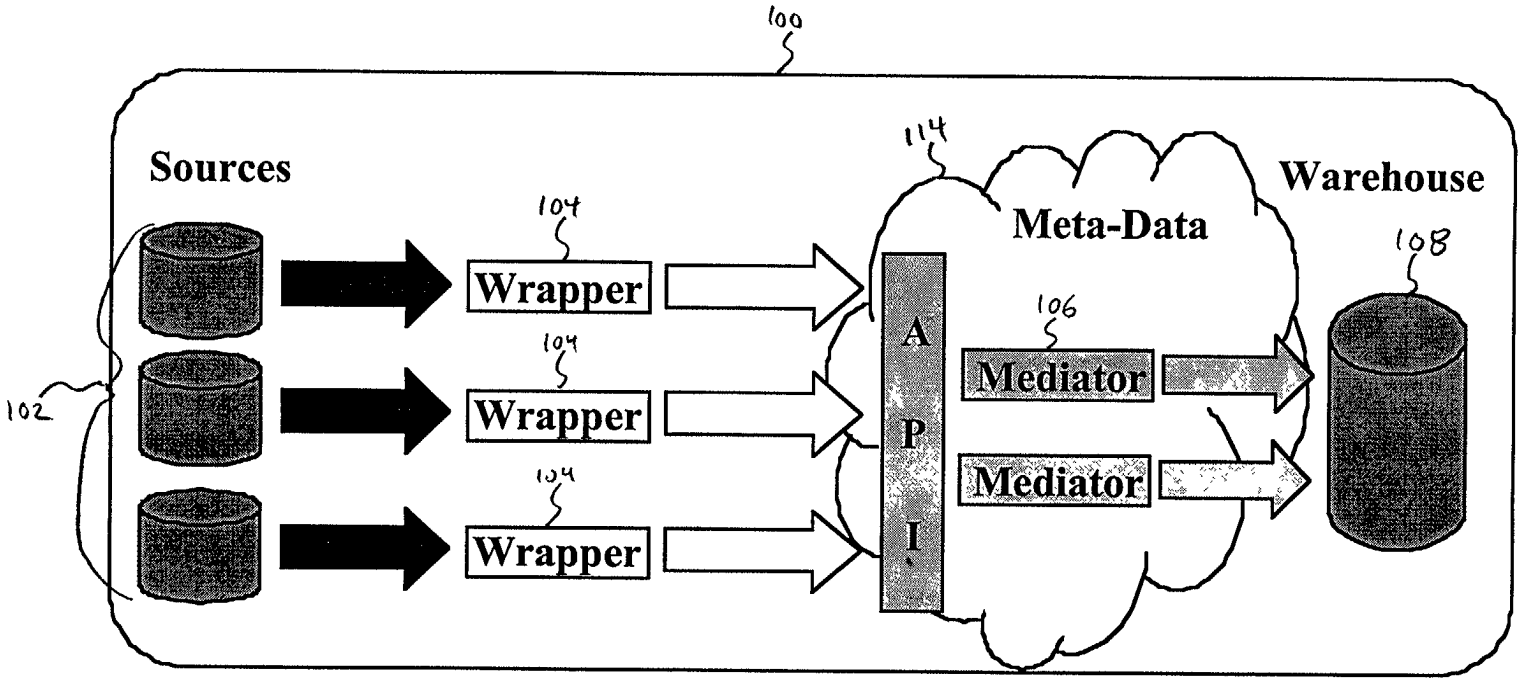
100
Sources
102
104
Wrapper
104
Wrapper
104
Wrapper
114
Meta-Data
A
P
I
106
Mediator
Mediator
Warehouse
108

FIG. 2

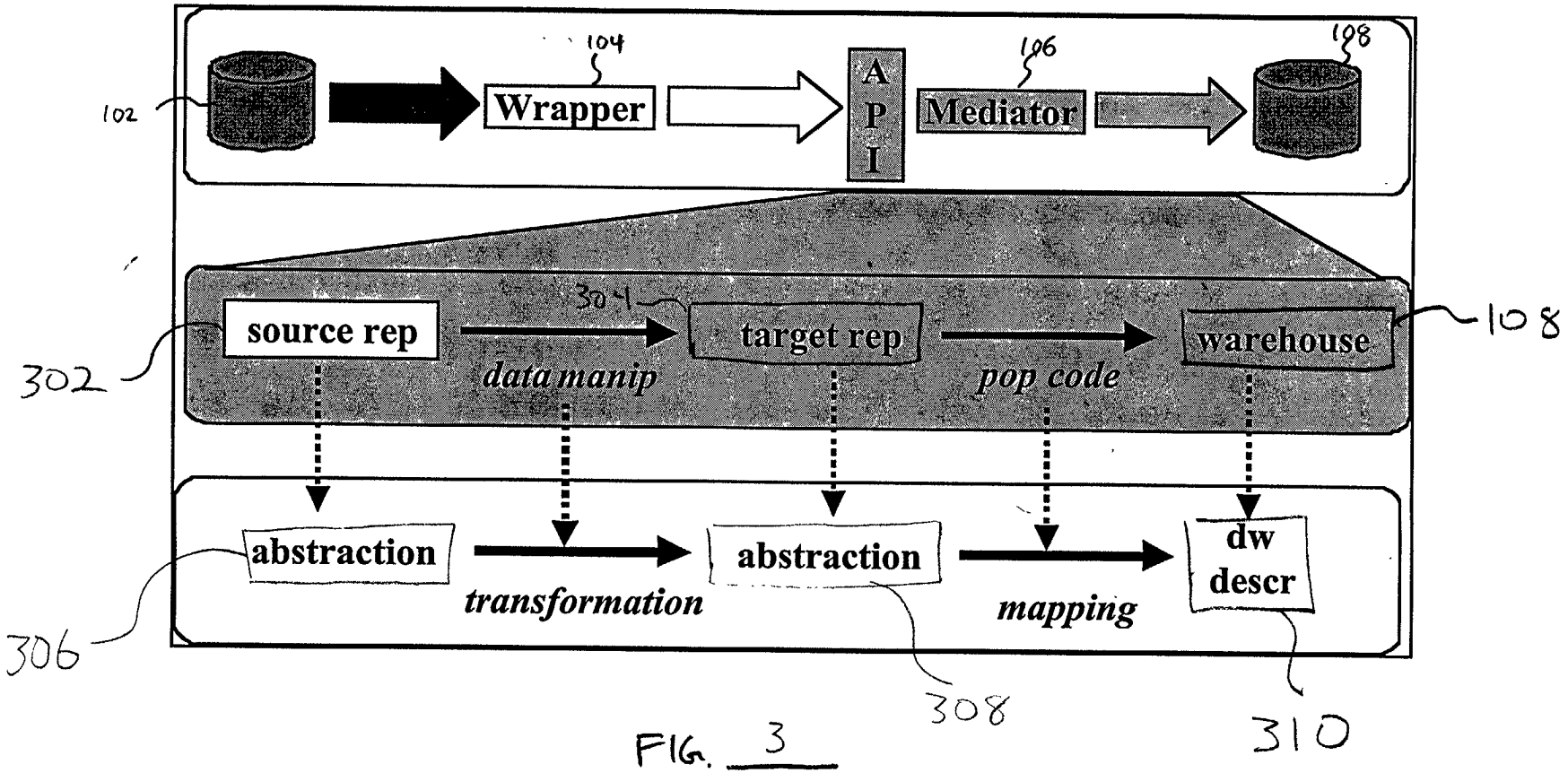
102
104
Wrapper
API
106
Mediator
108
source rep
302
304
data manip
target rep
pop code
warehouse
108
abstraction
306
transformation
abstraction
308
mapping
dw
descr
310

FIG. 3

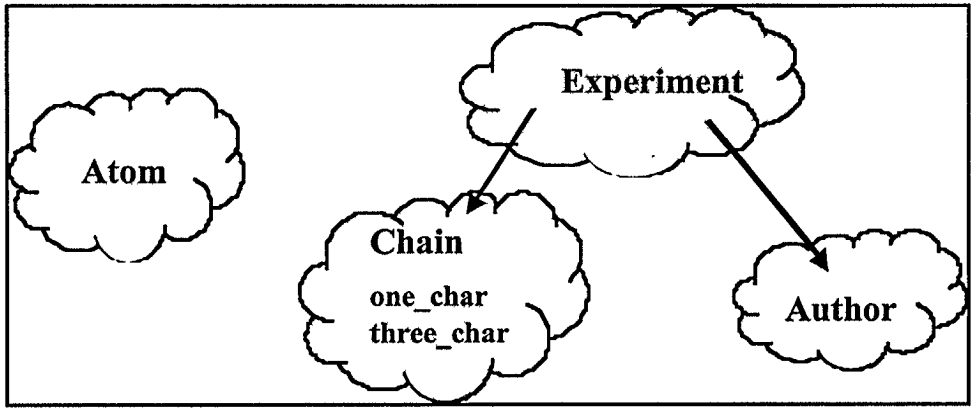

FIG. 4

09479432.010700

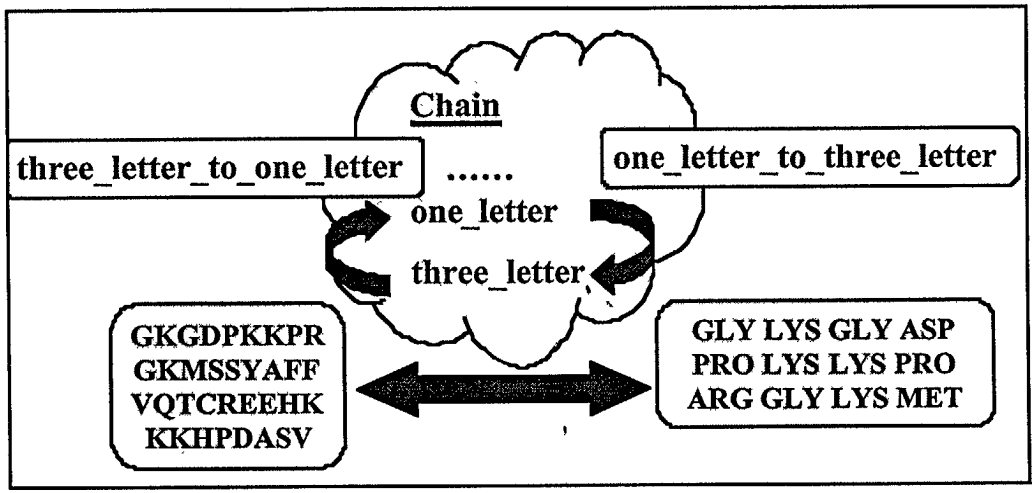

FIG. 5

09479432.030700

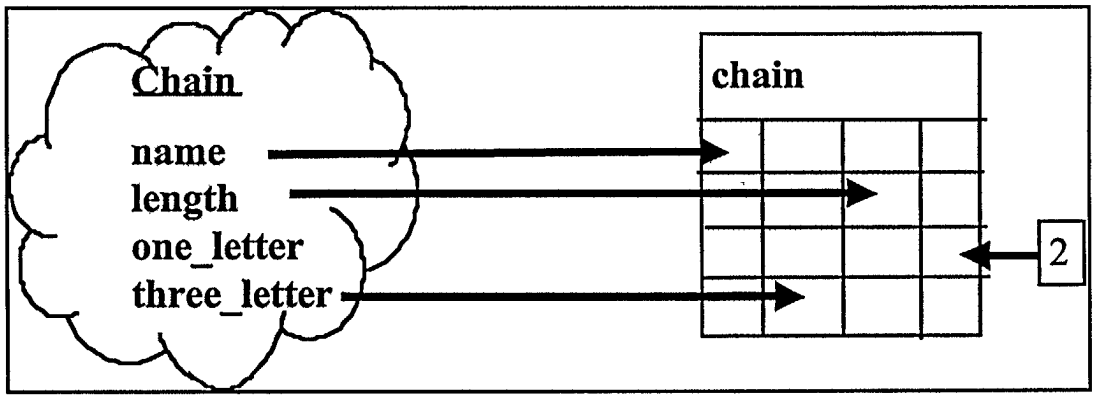

FIG. 6

09479432.010700

FIG. 7

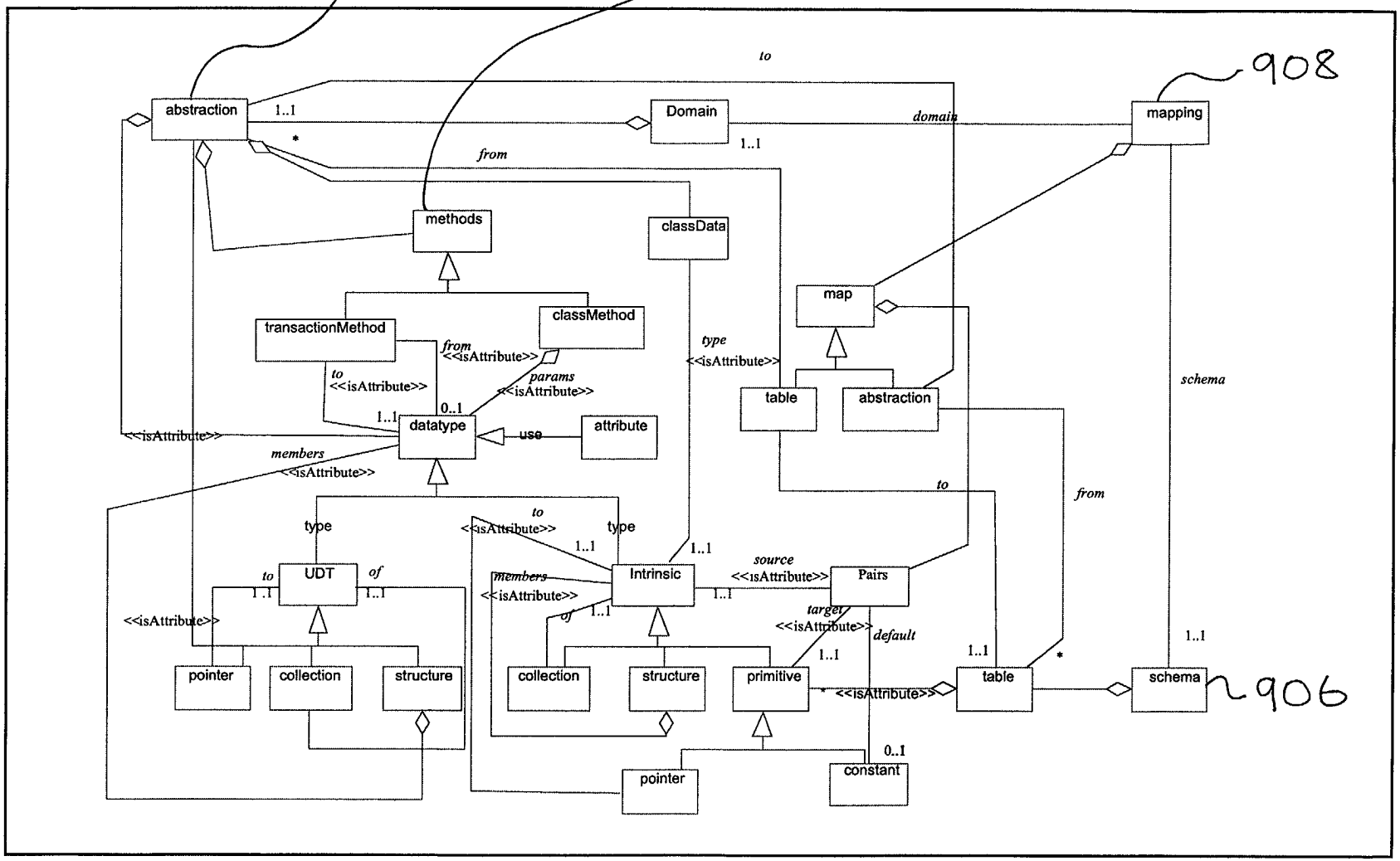

FIG. 8

09473432.010700

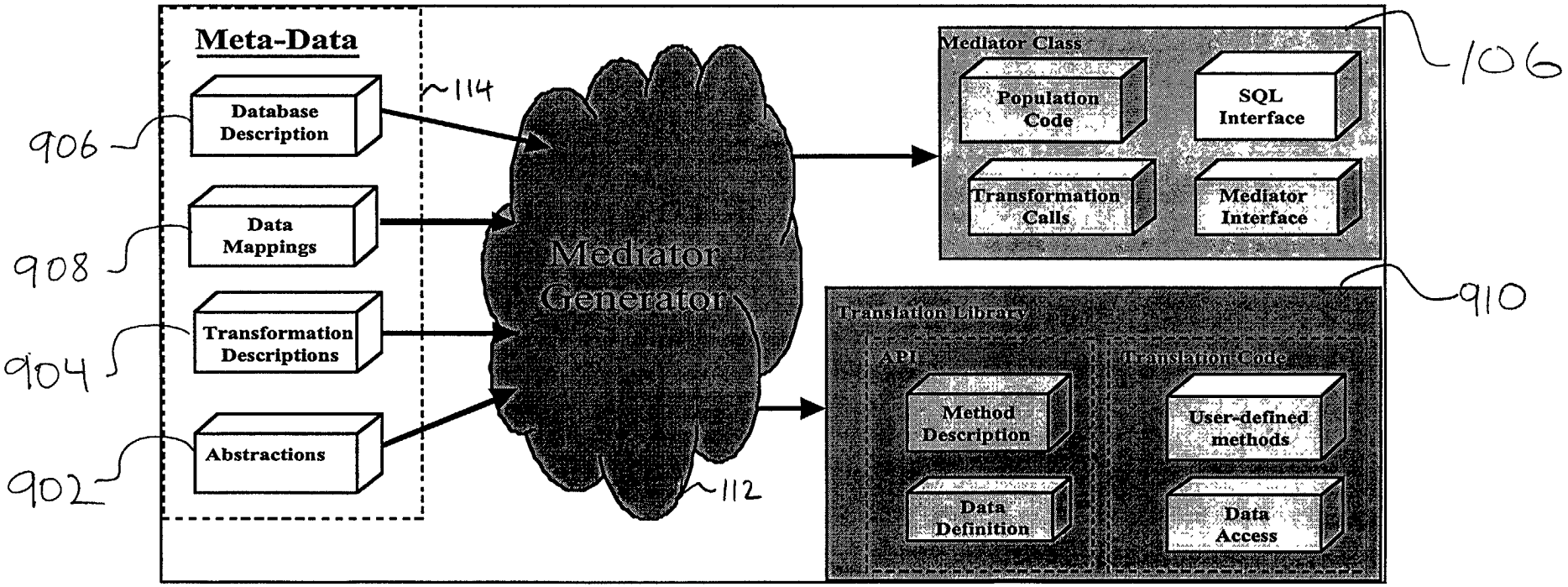

FIG. 9

09479432.010700

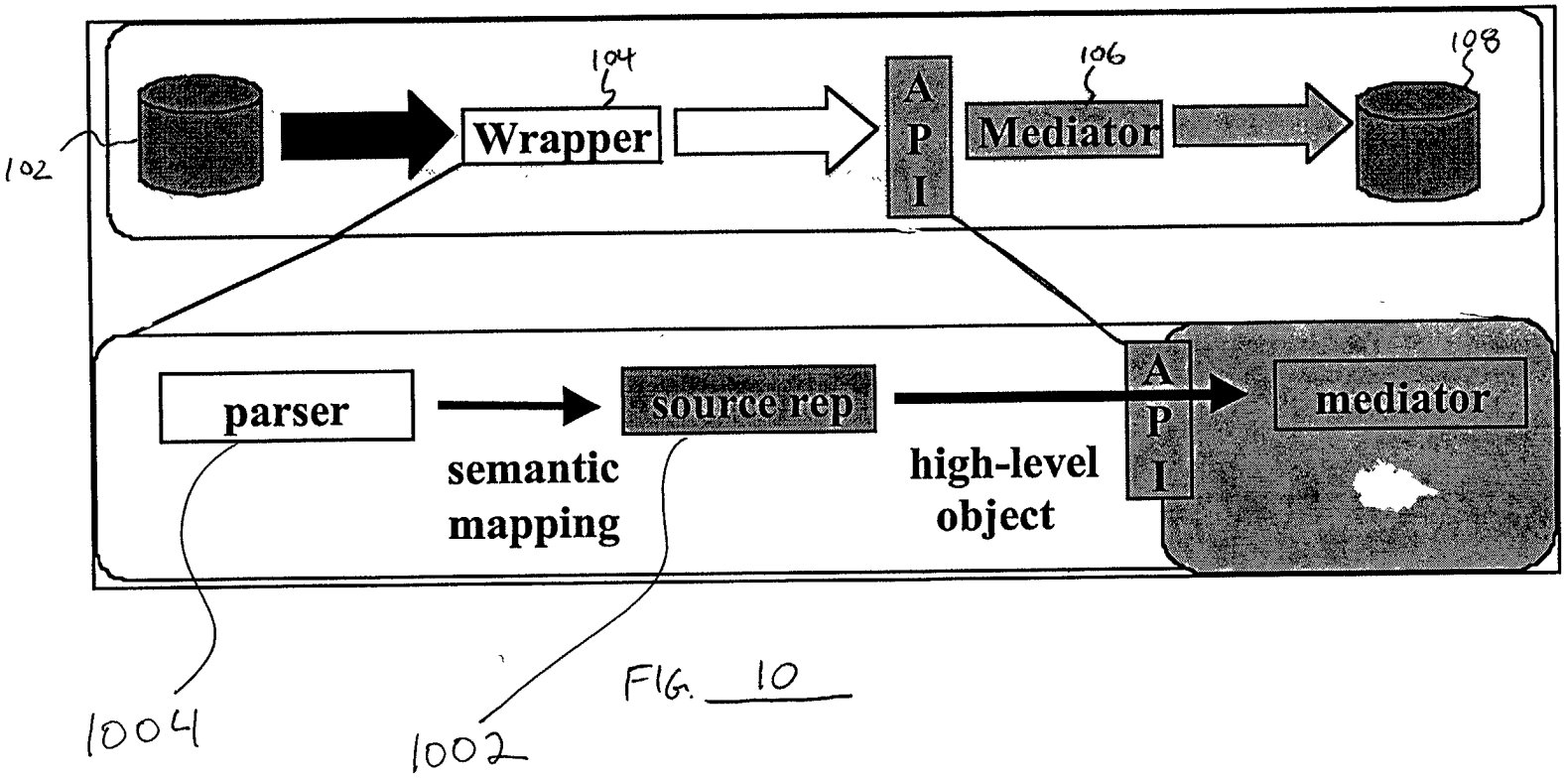

FIG. 10

09479432.030700